Benutzerdefinierte Datentypen

- Bisher wurden lediglich die primitiven Datentypen betrachtet.
- c gestattet, wie die meisten höheren Programmiersprachen, die Definition komplexerer Datentypen zur Modellierung von konkreten oder abstrakten Sachverhalten.

Datentyp struct

- Strukturen dienen der Modellierung von Sachverhalten.
- In Strukturen kann man Daten verschiedener anderer Datentypen, die in ihrer Gesamtheit einen Sachverhalt hinlänglich beschreiben, zusammenfassen.
- Um für ein Programm einen Studenten zu beschreiben, könnten
  - Name (Zeichenkette, char-Array),
  - Matrikelnummer (int-Wert),
  - Klausurnote (float-Wert),
  - Belegnote (int-Wert)

  von Interesse sein.

Datentyp struct

```
4   struct tStudent
5   {
6     char  name[30+1];
7     int   matrNr;
8     float noteKl;
9     int   noteBel;
10  };
```

Strukturkomponenten,
Sie haben einen eigenen Namensraum innerhalb der Struktur.

- Strukturvereinbarung für den Typ

  `struct tStudent`

- Der Typname besteht zunächst aus dem Schlüsselwort struct, gefolgt von einem frei wählbaren Namen.
- In den geschweiften Klammern sind die Komponenten definiert.

Datentyp struct

- Von Strukturdatenypen kann man Variablen anlegen und diese auch initialisieren.
- Die Initialisierung erfolgt durch eine Werteliste, wie bei Arrays.
- Die Werte in dieser Liste müssen zu den Komponententypen typverträglich sein.
- Enthält die Liste weniger Elemente, als die Struktur Komponenten hat, so wird mit Nullen aufgefüllt.

Datentyp struct

```
21  int main(int argc, char** argv)
22  {
23      struct tStudent stud={"Hans Huckebein", 12321};
24      displayStudent(stud);
25      return 0;
26  }
```

- In Zeile 23 wird eine Strukturvariable, man spricht mitunter auch von einem Strukturobjekt angelegt und initialisiert.
- In zeile 24 wird eine Funktion aufgerufen, die die Strukturvariable (als Kopie) als Parameter übergeben bekommt, sie soll die Werte der Strukturvariablen ausgeben.

Datentyp struct

- Auf die einzelnen Komponenten einer Stuktur kann man mittels des Punktselektors zugeriffen.
- Strukturen haben eine feste Länge im Speicher, sie soll **immer mit sizeof** ermittelt werden.
- Strukturen kann man einander zuweisen (Arrays nicht!)

```c
#include <stdio.h>
#include <stdlib.h>

struct tStudent
{
  char  name[30+1];
  int   matrNr;
  float noteKl;
  int   noteBel;
};

void displayStudent(struct tStudent s)
{
    printf("%-30s, %6d, %3.1f, %d\n",
           s.name,
           s.matrNr,
           s.noteKl,
           s.noteBel);
}

int main(int argc, char** argv)
{
    struct tStudent stud={"Hans Huckebein", 12321};
    displayStudent(stud);
    return 0;
}
```

Das komplette Beispiel, schreiben Sie es ab und probieren Sie es aus.

```
$ ./a.out
Hans Huckebein                 ,  12321, 0.0, 0
$
```

```c
int main(int argc, char** argv)
{
  if (argc !=3)
  {
      printf("usage: %s <name> <matrikelnr>\n",argv[0]);
      exit (-1);
  }
  struct tStudent stud;
  strcpy(stud.name,argv[1]);
  stud.matrNr=atoi(argv[2]);
  stud.noteBel=0;
  stud.noteKl =0;
  displayStudent(stud);
  return 0;
}
```

```
$ ./a.out 'hans huckebein' 12321
hans huckebein               ,  12321, 0.0, 0
$
```

Hier werden die Komponenten durch Kommandozeilenargumente gefüllt. Den Namen muss man mit strcpy kopieren, er kann nicht zugewiesen werden. Informieren Sie sich über strcpy (man-Pages, Internet ...)
Die Notenwerte müssen expliztit auf 0 gesetzt werden, sie bleiben sonst undefiniert (was dort eben zuletzt im Speicher stand).

typedef

- Die wiederholte Angabe von **`struct name`** ist etwas umständlich.
- Die Verwendung von typdef ist oft vorteilhaft.
- Typedef definiert einen neuen Datentyp aus einem vorhandenen Datentyp.

```
 5 typedef struct tStudent
 6 {
 7   char  name[30+1];
 8   int   matrNr;
 9   float noteKl;
10   int   noteBel;
11 }tStud;
```

Neuer Datentyp tStud. Der Name tStudent in der ersten Zeile Kann hier auch in vielen Fällen entfallen.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct tStudent
{
  char  name[30+1];
  int   matrNr;
  float noteKl;
  int   noteBel;
}tStud;

void displayStudent(tStud s)
{
    printf("%-30s, %6d, %3.1f, %d\n",
           s.name,
           s.matrNr,
           s.noteKl,
           s.noteBel);
}

int main(int argc, char** argv)
{
  struct tStudent stud={"Hans Huckebein", 12321};
  displayStudent(stud);
  return 0;
}
```

- In den Funktionen main und displayStudent wird nun nur noch der Typ tStud verwendet.
- Es ist synonym zu struct tStudent.

Datenlänge von Strukturen

- Die Datenlänge soll immer mittels **sizeof** ermittelt werden.
- Addiert man die Längen (31+4+4+4) erhält man den Wert 43.
- Mittels sizeof erhält man den Wert 44.
- Der Grund liegt in Bytes, die aus Gründen der Zugriffsoptimierung vom Compiler eingefügt werden, um Komponenten auf einer z.Bsp.: durch 4 teilbaren Adresse beginnen zu lassen (bezeichnet als Alignment).

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct tStudent
{
  char  name[30+1];
  int   matrNr;
  float noteKl;
  int   noteBel;
}tStud;



int main(int argc, char** argv)
{
    tStud stud={"Hans Huckebein", 12321};
    printf("Length of stud: %ld\n",sizeof stud);
  return 0;
}
```

Auf einer anderen Plattform, anderes Betriebssystem, andere Rechnerhardware (Raspberry oder Microcontroller) erhalten Sie möglicher Weise andere Werte.

Pointer auf Strukturen

- Es können auch Pointer, die auf Strukturen verweisen, definiert werden.
- Dies geschieht in der gewohnten Form:

```
struct tStudent * pStud;     // ohne typedef
tStud * pStud;               // mit typedef
```

- Um über einen Pointer auf die Strukturkomponenten zuzugreifen, verwendet man den Pfeiloperator ->.
- Die Funktion displayStudent könnte wie folgt geändert werden:

```
13  void displayStudent(tStud* s)
14  {
15      printf("%-30s, %6d, %3.1f, %d\n",
16              s->name,
17              s->matrNr,
18              s->noteKl,
19              s->noteBel);
20  }
21
22   . . .
23
24    displayStudent(&stud);
```

Verwendung des Operators ->
Weil ein Pointer auf eine Struktur vorliegt.

In main

- Hier wird an Stelle einer Kopie der ganzen Struktur nur die Adresse der Struktur als Parameter übergeben.
- Das ist sehr viel effizienter in Bezug auf den Speicherbedarf als auch in Bezug auf die Laufzeit.
- Aber Achtung: Änderungen, die eine Funktion in dieser Konstellation an der Struktur vornimmt, wirken sich auf das Originalobjekt aus.
- Die Strukturkomponenten werden hier durch den -> Operator selektiert.

Arrays von Strukturen

- Man kann auch Arrays von Strukturen bauen.
- Arrays von Strukturen können ebenfalls initialisiert werden.
- Auch hier assoziiert der Arrayname die Adresse des ersten Elements.
- Auch hier kann man einen Pointer auf das Array einrichten und mit Pointerarithemtik durch das Array navigieren.
- Ein Increment auf einen solchen Pointer bewirkt das Weiterstellen des Pointers auf die nächste Struktur im Array.

Arrays von Strukturen

```
22  int main(int argc, char** argv)
23  {
24    struct tStudent students[]={{"Hans Huckebein", 12321},
25                                {"Peter Lustig", 23123},
26                                {"Anna Dunzinger",32545}};
27    int i;
28    for (i=0; i<sizeof students/sizeof (tStud); i++)
29      displayStudent(students+i);
30    return 0;
31  }
```

Da students die Adresse auf die erste Struktur repräsentiert, können wir mit students+i, das ist hier Pointerarithmetik, das Array durchlaufen.

Bitfelder

```c
#include <stdio.h>

int main()
{
  struct
  {
    unsigned int a:3;
    unsigned int b:3;
    int c:8;
  }st={5,7,99};
  printf("sizeof: %ld a:%3d, b:%3d, c:%d\n",
         sizeof   st,st.a,  st.b,  st.c);
  return 0;
}
```

Bitfelder gestatten es, innerhalb von Strukturen kleine Int-Werte in einem Integerwert zusammenzufassen. Die Zahl hinter dem Doppelpunkt nach dem Komponentennamen gibt die Anzahl der Bits, die diese Komponente belegen soll, an. Im weiteren erfolgt die Benutzung wie bei gewöhnlichen Strukturkomponenten.

Unions

- Unions sind ebenfalls benutzerdefinierte Datentypen.
- Sie ähnlen syntaktisch den Strukturen.
- Sie werden durch das Schlüsselwort union eingeleitet und führen eine Reihe von Komponenten unterschiedlichen Datentyps ein.

Unions

- Im Unterschied zu Strukturen liegen die Komponenten eines Unions alle auf der selben Adresse. Sie überlappen einander ab dem ersten Byte.
- Unions ermöglichen, ein und dieselbe Bitfolge durch verschiedene Datentypen unterschiedlich zu interpretieren.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef union
{
    int i;
    unsigned char array[sizeof (int)];
    float f;
} tdata;

int main(int argc, char** argv)
{
   int i;
   tdata data;
   float f=atof(argv[1]);
   data.f=f;
   printf("data.i (int)      : %d\n",data.i);
   printf("data.f (float)    : %f\n",data.f);
   printf("data.array        : ");
   for(i=0; i<sizeof (int);i++)
       printf("%02x ",data.array[i]);
   printf("\n");
   return 0;
}
```

Unions

- Im Beispiel wird eine float-zahl eigegeben.
- Ihr Bitmuster wird unter dem Namen i als int-Wert interprtiert.
- Das selbe Bitmuster wird unter dem Namen array als 4 Bytes interpretiert.

Unions

- Bei der Modellierung von Daten werden unions auch zur Modellierung unterschiedlicher Klassen von Sachverhalten genutzt.
- Betrachten wir PKW und LKW als Fahrzeuge,
- So sind PKW durch die Anzahl zugel. Personen, LKW durch die Zuladung gekennzeichnet.
- Über einKennzeichen (hier typ) wird gesteuert, weches Element des Unions benutzt werden soll.

```
 5  typedef struct
 6  {
 7      char marke[16+1];
 8      char typ; // l:lkw, p: pkw
 9      union
10      {
11          int    zulPers;
12          float  zulLadung;
13      }beschreibung;
14  } tfahrzeug;
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct
{
   char marke[16+1];
   char typ; // l:lkw, p: pkw
   union
   {
       int    zulPers;
       float  zulLadung;
   }beschreibung;
} tfahrzeug;

int main(int argc, char** argv)
{
   tfahrzeug f[]={{"opel",'p',5}, // 5 Personen
                  {"vw",'l'}};    // 3.5 Tonnen ZulLadung
   f[1].beschreibung.zulLadung=3.5;
   int i;
   for (i=0; i< sizeof f/sizeof (tfahrzeug); i++)
       if (f[i].typ=='p') printf("%-5s: %4d Personen\n",f[i].marke, f[i].beschreibung.zulPers);   else
       if (f[i].typ=='l') printf("%-5s: %4.1f Ladung\n",f[i].marke, f[i].beschreibung.zulLadung); else

   return 0;
}
```

Unions werden bei der Initialiserung mit einem Wert der ersten Komponente Initialisiert (hier int).
Der Floatwert muss über eine Zuweisung explizit eingetragen werden.

Enumerations

- Der Dateyp enum führt eine Liste von benannten, aufeinanderfolgenden int-Werten ein.
- Standardmäßig beginnen die Werte mit 0.
- Innerhalb der Liste können auch Werte zugeordnet werden, die weiteren nachfolgenden Werte steigen dann ab da jeweils um 1.
- Werte einer enumeration sind typkompatibel mit int.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

enum wota{montag,
          dienstag,
          mittwoch,
          donnerstag,
          freitag,
          samstag,
          sonntag};


int main(int argc, char** argv)
{
   switch(argv[1][0]-'0')
   {
        case montag    :
        case dienstag  :
        case mittwoch  :
        case donnerstag:
        case freitag   :
            puts("Arbeitstag"); break;
        case samstag   :
        case sonntag   :
            puts("frei"); break;
   }
   return 0;
}
```

Macht aus ascii-Zeichen eine int-Zahl 0x33-0x30 → 3

- Im Beispiel werden unter den Namen montag bis sonntag int-Konstanten mit den Werten 0..6 vereinbart.
- main übernimmt eine Ziffer als Kommandozeilenargument.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

enum wota{montag    ,
          dienstag  ,
          mittwoch  ,
          donnerstag,
          freitag   ,
          samstag   =freitag+16+1,
          sonntag   };


int main(int argc, char** argv)
{
  printf("%02x, ",montag);
  printf("%02x, ",dienstag  );
  printf("%02x, ",mittwoch  );
  printf("%02x, ",donnerstag);
  printf("%02x, ",freitag   );
  printf("%02x, ",samstag   );
  printf("%02x, ",sonntag   );
  printf("\n");
}
```

- Hier wird in Zeile 10 die Nummerierung durchbrochen.
- In samstag wird das bit 2^4 gesetzt.
- Auch bei sonntag ist dieses Bit als kennzeichen für arbeitsfreien Tag gesetzt.

```
$ ./a.out
00, 01, 02, 03, 04, 15, 16,
$
```